IT UNIVERSITY OF COPENHAGEN

Machine Learning Project

Machine Learning

BSMALEA1KU

Anna Maria Gnat (agna@itu.dk)
Josefine Fritz Nyeng (jnye@itu.dk)
Pedro Nuno Castanho Prazeres (peca@itu.dk)

BSc in Data Science
IT University of Copenhagen
January, 2024

Contents

1 Introduction

The goal of this report is to summarise the process and the findings of a project we did as a part of our course in Machine Learning at IT University of Copenhagen. We were tasked with solving a classification problem on a famous fashion-MNIST dataset. We investigated the dataset itself and then proceeded to implement different machine-learning algorithms. Some of the implemented algorithms were developed with only the use of math-related Python libraries, for a more in-depth understanding. For others, we used well-established algorithms from libraries such as scikit-learn or PyTorch. After analysing the performance of the algorithms, we conclude our findings in the final section of this report.

1.1 Data set

For this project, we were provided with a sample from the fashion-MNIST dataset. It comprises 15,000 gray scale images, each of a size 28x28 pixels. The images come from the Zalando website and are labelled according to the type of clothing they represent. Table 1 shows the labels and the corresponding classes of pieces of clothing.

Label	Type of clothing
0	T-shirt/Top
1	Trouser
2	Pullover
3	Dress
4	Shirt

Table 1: Labels and the corresponding categories of clothing

The set we received was pre-divided into training and test sets, with 10,000 images in the training set and 5,000 images in the test set. The data was provided to us as a numpy array. The first 784 columns represent pixel values for each pixel, and the last column indicates the label of the clothing item visible in the image.

1.2 Problem

The main problem of this project was developing machine-learning algorithms that would classify the images as certain pieces of clothing with the best performance possible, according to various metrics. We also set out to study and compare two methods of dimensionality reduction. Lastly, by implementing some of the algorithms from scratch - LDA and Naive Bayes - we need to study them thoroughly, allowing us to gain an in-depth understanding of how they work and their applicability to the fashion-MNIST dataset. By comparing our custom implementations with the established libraries' algorithm we hope to gain insights into the trade-off between simplicity and performance. Throughout this project, our aim is not only to achieve high classification accuracy but also to contribute to our understanding of the strengths and limitations of different machine-learning approaches in the context of image classification.

2 Exploratory Data Analysis

2.1 Preprocessing

As a prepossessing step, the pixel values of all the images were scaled to have a mean of 0 and a variance of 1 using the StandardScaler from the scikit-learn library [skl, 2023]. This was done to allow the machine-learning models to interpret the pixel features of each image on the same scale. Without the scaling, variations in the feature values could lead to issues during the learning process and could cause bias in the classifiers. Furthermore,

principal component analysis is also very sensitive to the scales of the variables. Scaling the data helps ensure that all features contribute equally to the analysis. Without scaling, the features with high variance simply due to their scales would dominate in the direction of the principal components. Thus, scaling the features allows PCA to focus on capturing the direction of the greatest variance in the data. The scaled training data was used for the remainder of this project.

2.2 Dimensionality reduction

Dimensionality reduction is the process of transforming data from a high-dimensional space to a lower-dimensional space while preserving the meaningful properties of the original data. This helps eliminate noise and redundant features as well as limits overfitting, which can occur when there are too many features.

2.2.1 PCA

Principal component analysis, PCA, is an unsupervised method used for dimensionality reduction. PCA transforms and combines the original features into a smaller set of principal components that point in the direction of maximum variance in the data. The principal components are computed by finding the eigenvectors of the covariance matrix of the data. The eigenvectors are ordered such that the first eigenvector corresponds to the first principal component and captures the maximum amount of variance and so on. The data is then projected onto this new set of principal components, thus transforming the feature space into a lower dimension. Each image has 784 pixel values which function as features. Using PCA this feature space was reduced to the first 2 principal components. The projection of the data onto these principal components is shown in Figure 1 below.

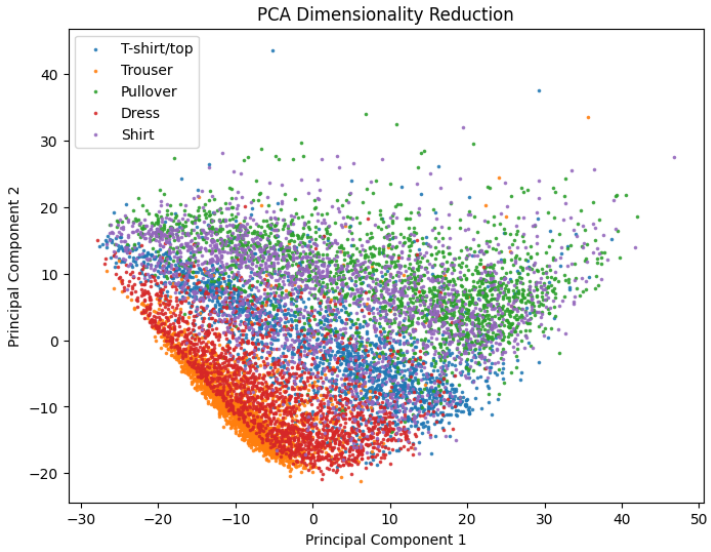

Figure 1: Projection onto the first 2 principal components

Since PCA is an unsupervised method, it has no knowledge of the classes in the data. Therefore, while the first 2 principal components capture the directions of greatest variance, they are not particularly good at separating the classes. This can be seen in Figure 1 as there is quite a lot of overlap between the classes.

To interpret these principal components the first 4 eigenvectors representing the first 4 principal components have been visualised below in Figure 2. These visualisations represent the most prominent patterns captured by each principal component. The seismic colormap has been applied which shows negative values as blue, positive values as red, and 0 values as white. When moving 1 unit in the direction of a principal component, the

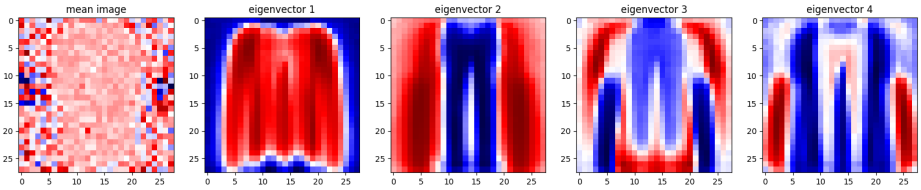

Figure 2: Visualisation of the first 4 principal components

corresponding eigenvector is added to the image. Thus, the negative values will subtract from the image and the positive values will add to the image. The first eigenvector has positive values in the shape of a pullover which could indicate that this principal component is good at capturing the features or patterns associated with pullovers and is thus good at separating this class from the others. Similarly, eigenvector 2 seems to have an area of negative values in the shape of trousers, indicating that this principal component captures features that differentiate the Trousers class from other classes. Eigenvectors 3 and 4 seem to capture the presence of sleeves but otherwise, their patterns are harder to distinguish in terms of classes. This is expected since PCA is unsupervised and thus does not focus on class separability.

Principal components can also be used to approximate the original data. Each image can be reconstructed using the first m principal components which approximates the image in its original high-dimensional space. This is shown for different values of m in Figure 3 below. As the number of principal components m increases, the reconstruction becomes

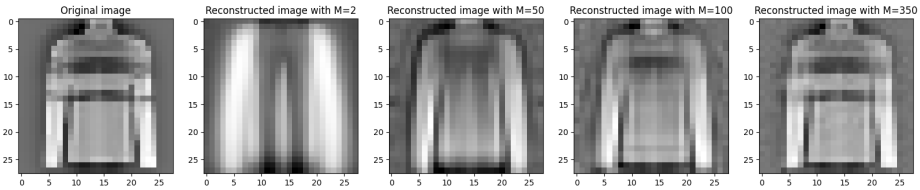

Figure 3: Reconstructed images

more accurate. This can be seen in Figure 3 as more and more details of the original image are accurately captured as m increases. Eventually, when all the principal components are used, $m = 784$, the image will be an exact reconstruction of the original image.

2.2.2 LDA

Linear discriminant analysis, LDA, is a supervised technique for feature space reduction and class separation. It models the joint probability distribution $p(x, k)$ and uses this to estimate posterior class probabilities $P(Y = k|x)$, with the goal of finding linear combinations of features that separate classes as much as possible. In the multi-class classification problem, this separation is achieved by finding the matrix W. It is the matrix that contains the linear discriminant variables later used for reducing the dimension of the feature space. This matrix, W, and thus, the maximal separation between the classes, is achieved by maximising:

$$J(W) = \frac{|W^T S_B W|}{|W^T S_B W|} \tag{1}$$

where S_W is the within-class scatter matrix, and S_B is the between-class scatter matrix, given by the following equations:

$$S_W = \sum_{i=1}^{k} (x_i - \mu_i)(x_i - \mu_i)^T \tag{2}$$

where x_i is each observation for each class and μ_i is the class mean.

$$S_B = \sum_{i=1}^{k} N_i(\mu_i - \mu)(\mu_i - \mu)^T \tag{3}$$

where N_i is the number of observations in class i, μ_i is the class mean, and μ is the overall central point.

We developed a custom LDA class to analyse the fashion-MNIST dataset using Python and the NumPy library. It is a close implementation of the mathematical calculations [Alpaydin, 2014] that were made available to us during our coursework. The class initialises with the variables `X` and `y` which are the training images and their labels respectively. The LDA class also computes the class means and the overall central point, which is the average of all the class means, using NumPy's `mean`. The scatter matrices, S_W and S_B, were computed using Equations 2 and 3.

The matrix that is derived from multiplying the inverse of S_W with S_B, has the property that its eigenvectors are the linear discriminant variables that make up the matrix W that we are looking for. The eigenvalues and eigenvectors are computed with NumPy's `linalg.eig`. The eigenvectors corresponding to the n largest eigenvalues are selected for the new feature space. We implemented a function, `main_lds`, which returns the matrix, W, whose columns make up the n largest eigenvectors i.e. the linear discriminant variables. This matrix is then used for the projection of the data onto the lower-dimensional space.

The function `projection` returns the projection of the data X onto the new subspace by multiplying it with the projection matrix returned by `main_lds`, W, therefore reducing its dimensionality:

$$X_{lda} = XW$$

In our case, we decided to use the first $n = 2$ linear discriminant variables. Therefore, this projection results in a matrix with each observation along the rows and the values for the two linear discriminant variables along the columns.

To assert the correctness of our implementation we applied both our LDA implementation and scikit-learn's version to the provided dataset in order to compare.
As shown in Figure 4, the LDA projections of both methods are visually similar, with differences only on scale. There were no differences in the values for the class means, the central point, and within-class matrices. However, the between-class matrices of the two implementations differ slightly, due to the scaling applied by sklearn. This should not, however, significantly affect the performance of either implementation when compared to the other.

Figure 4: Comparison between projections of the two LDA implementations

Note that in order to use the linear discriminant variables later for the Bayes classifier, an additional function, `transform`, was added. This function accepts input data in the

original high-dimensional space and projects it onto the main linear discriminants by using the same method as `projection`. Thus, returning the linear discriminant variables for the input data.

2.2.3 PCA vs LDA

The key distinction between PCA and LDA lies in the way they create new features. PCA, being an unsupervised method, does not consider class labels. It creates new features, known as principal components, based only on the variance in the data. This might be good for some cases, but might not be optimal when it comes to discrimination of the classes. This is evident in Figure 5, where significant class overlap can be observed in the PCA-reduced space.

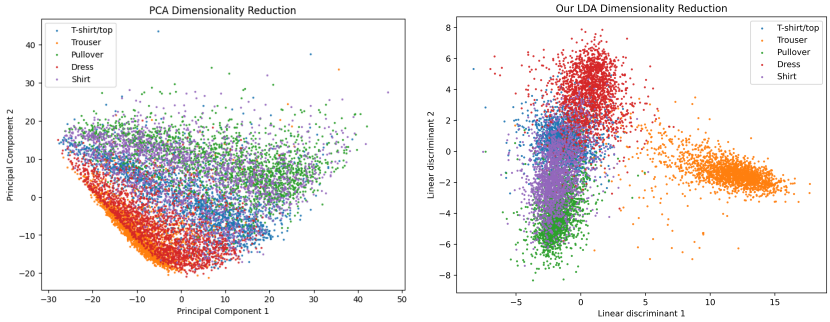

Figure 5: Comparing the PCA and LDA plots

On the contrary, LDA, as a supervised technique, seeks to maximise class separation. The LDA scatter plot in Figure 5 demonstrates clearer boundaries between different classes. Specifically, the Dress and Trouser classes are very well separated from the others. The Shirt class seems to overlap quite a bit with the T-shirt/top and Pullover classes. Intuitively this makes sense since these categories are quite similar. However, overall we can see that the linear discriminant variables are better at separating the classes than the principal components. Therefore, despite PCA's utility in capturing maximum variance, LDA's ability to use label information for maximising class separation likely results in a better-suited feature space for classification purposes.

3 Classification

To achieve the goals of this project, we needed to implement, train and evaluate three different machine-learning models. The classifiers were all run through cross-validation during training, to validate their performance and flush out any potential issues. Then we saved the trained models to use them for predictions on the test set.

3.1 Naive Bayes

The Naive Bayes classifier is based on Bayes Theorem, which is a formula for deriving the posterior probability of an observation being in a certain class, k, given its features, x_1, x_2:

$$p(Y = k|x_1, x_2) = \frac{p(x_1, x_2|Y = k)p(Y = k)}{p(x_1, x_2)} \tag{4}$$

where the features used for our implementation, x_1, x_2, are the first two linear discriminant variables. The classifier then assigns each input image to the class with the highest posterior probability. The formula involves the class priors, $\pi_k = P(Y = k)$, and the class conditionals, $f(x_1, x_2|k) = P(x_1, x_2|Y = k)$, which are essentially the joint multivariate distribution of the features, x_1, x_2 given a class, k. Naive Bayes assumes that the features

within each class are independent of each other. This means that the joint multivariate distribution of the features given a class, k, can be modeled as the product of the marginal univariate distributions for each feature given that class:

$$f(x_1, x_2|Y = k) = f(x_1|Y = k)f(x_2|Y = k) \tag{5}$$

The probability, $p(x_1, x_2)$, of observing the features regardless of the class can then be estimated as the sum of the class prior multiplied by the class conditional for each possible class:

$$p(x) = \sum_{k=1}^{K} \pi_k f(x_1, x_2|Y = k) \tag{6}$$

Thus for our classification setting, the formula for the posterior probability can be rewritten as follows:

$$p(Y = k|x_1, x_2) = \frac{\pi_k f(x_1, x_2|Y = k)}{\sum_{k=1}^{K} \pi_k f(x_1, x_2|Y = k)} \tag{7}$$

Now, to compute the posterior probability using Bayes theorem, the classifier must estimate the class priors π_k and the univariate feature distributions, $f(x_1|Y = k)$ and $f(x_2|Y = k)$. The class priors were estimated as the relative frequency of each class in the training data. The univariate feature distributions were estimated using kernel density estimation (KDE), which is a non-parametric method used to estimate probability distributions (pdf) of a set of observations. For our implementation, we decided to apply KDE using the Gaussian Kernel, $K(x)$:

$$K(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp -\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2} \tag{8}$$

where the standard deviation, σ, is assigned to be the bandwidth, and the mean, μ, is assigned to be each training observation. This means that the Gaussian Kernel fits a Gaussian bell curve around each observation of a feature x_i in each class k. Then all of these bell curves are summed together to estimate the pdf of that feature for that class i.e. the univariate feature distribution:

$$f(x|Y = k) = \frac{1}{nh} \sum_{i=1}^{n} K(\frac{x - x_i}{h}) \tag{9}$$

In this formula, n is the number of training observations in class k and h is the bandwidth. For our implementation, we selected the bandwidth using Scott's rule of thumb [Liu et al., 2009]:

$$h = 1.06 \times \hat{\sigma} n^{-1/5}$$

This formula approximates the appropriate bandwidth using the sample standard deviation of the training data, $\hat{\sigma}$, and the number of training observations, n. For our training data, the resulting bandwidth was $h = 0.1679986784008780$.

After computing all the univariate feature distributions using KDE (equations 8 and 9), we multiplied them together for each class to obtain the multivariate feature distributions, $f(x_1, x_2|Y = k)$ using equation 5. These, along with the estimates for the class priors, were then substituted into formula 7 to compute the posterior probability for each class. Lastly, the classifier then classifies each input observation to the class with the highest posterior probability. Our code implementation is simply these formulas written as Python functions. We assessed the correctness of each function by comparing its output for a given test observation to the manual computation done by hand.

3.2 Support Vector Machine

Support vector machines are a set of supervised learning methods that can be used for both classification and regression problems. In addition to linear classification, they can

perform a non-linear classification, with the so-called kernel trick - using a kernel function on the original set of features to map them onto a higher-dimensional space.

SVMs are binary classifiers, but they can be used for multi-class problems in one of two ways: one-versus-one or one-versus-all. In the first method, an SVM model is trained on all possible pairs of classes and then returns the final class based on the amount of "votes" that were returned by each pairing. In the second method, one-versus-all, an SVM model is trained for each class, distinguishing that class from the rest of the observations. Each model then returns a probability of the new observation belonging to the specified class, and the class with the highest probability is the predicted class for that observation. Each method has advantages and disadvantages, for example, change of behaviour when there is an imbalance between the classes. However, for this project, the one-versus-one method was used as that is the implementation available through the scikit-learn library.

The basis of the SVMs is a maximal margin classifier. Its main idea is to create a hyperplane that separates the two classes with the widest margin possible. The hard margin classifier comes with some disadvantages that prevent us from using it in our classification problem. It depends on the total separability of the two classes - no misclassifications or overlaps are allowed. Therefore, we needed to use the soft margin SVM, a generalisation of the maximal margin classifier that extends to cases where the classes do overlap.

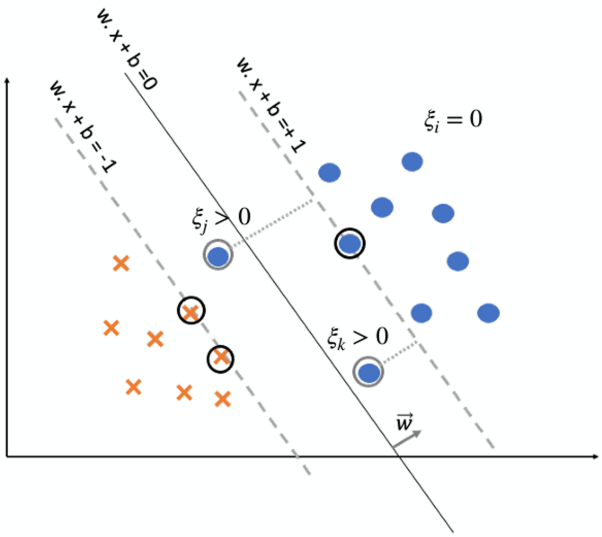

Figure 6: Visualisation of the soft-margin SVM [Zahadat, 2023]

The soft-margin classifier, also known as a support vector classifier or SVC, allows for some misclassifications, e.g. of the outliers, while providing a better classification overall. To allow this to happen, the soft SVM has an extra variable - a "slack variable" usually denoted as ξ. The slack variable is used as "penalty points" for overlapped and thus misclassified observations. Figure 6 visualises the soft-margin SVM. In the case of SVC, the classification becomes an optimisation problem where we need to maximise the width of the margin while minimising the penalty points - mistakes in classification. It is given by the equation:

$$\min_{(w,b)} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i \tag{10}$$

$$\text{subject to constraints:} \quad y_i(w \cdot x_i + b) \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0$$

From this, we can derive a dual representation of the optimisation problem using the Lagrange multipliers, which will allow us to represent it based only on the dot product of the observations. For the full derivation process, please see Appendix B. The dual representation is given by:

$$\max_{\alpha} \sum_{i}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) \tag{11}$$

$$\text{subject to } 0 \leq \alpha \leq C \quad \text{and} \quad \sum_{i}^{n} \alpha_i y_i = 0$$

In the soft SVM, the vector of Lagrange multipliers, α, has two bounds: a lower bound equal to 0 and an upper bound equal to C. The parameter C is the trade-off between achieving a low training error and a large margin. In other words, it balances the width of the margin and the amount of penalty points we are willing to accept. Setting C to be very large will make a soft SVM behave like a hard SVM.

To make the SVC even better, we can use a non-linear extension of the soft-margin SVM. It allows for the enlargement of the feature space using kernels to accommodate a non-linear boundary between classes. A non-linear SVM works by applying a transformation that brings the original feature space to a higher dimension, where the decision boundary is linear and then projects it back to the original feature space. While those transformations create complex, non-linear decision boundaries, the process in a high-dimension feature space requires a lot of computations, which can significantly slow down the process. This is where the so-called kernel trick can be used to speed this process up. Instead of performing the time-consuming operations of doing a transformation, we can use one of the kernel functions to transform the input data into a higher-dimensional space without the need to explicitly compute the transformation. There are many predefined kernel functions, but the most popular ones are: linear, polynomial and radial.

For our implementation of the SVM classifier on the fashion-MNIST dataset with the use of the scikit-learn library, we needed to establish the parameter C and choose the kernel function. We did this by looking at already established benchmarks for various algorithms [Xiao et al., 2017]. The best-performing SVC has the hyperparameter C of the value of 10 and uses the radial basis function as the kernel function. These are also the parameters we set for our SVM model in the code. If these were not available, we would perform a cross-validation to establish the best parameter value and choose the best kernel function.

3.3 Convolution Neural Network

Convolutional Neural Networks (CNNs) are a specialised form of neural networks for processing data with a grid-like topology, such as images. CNNs excel in capturing spatial features and patterns in data, making them particularly effective for image classification tasks [Chen et al., 2021]. They are characterised by their use of convolutional layers, which automatically and adaptively learn spatial hierarchies of features from input images.

Our CNN implementation for the fashion-MNIST dataset was developed using the PyTorch library [pyt, 2023], which also provided us with the ability to use CUDA for accelerated computation, enhancing both training and classification processes.

The architecture of our CNN includes several key components:

- **Convolutional Layers**: these apply a set of learnable filters to the input data, effectively capturing features such as edges, textures, and more complex patterns in the images.

- **ReLU Activation**: the Rectified Linear Unit (ReLU) activation function introduces non-linearity to the CNN, enabling it to learn complex relationships in the data.
- **Max Pooling**: this layer reduces the spatial dimensions of the input for the next convolutional layer, decreasing the number of parameters and computation in the network, which helps control overfitting.
- **Dropout Layers**: also to prevent overfitting, dropout layers randomly ignore selected neurons during training.
- **Fully Connected Layers**: these synthesise the learned features from previous layers to make final classification decisions.

The output of the network is finally passed through a log softmax layer, which is suitable for multi-class classification tasks since, on an assumption of Gaussian within-class distributions, it outputs a vector of posterior distributions per class [Bridle, 1990].
The CNN classifier then classifies the observations to the class with the highest posterior probability. A more detailed description and summary of the different building blocks of the implemented CNN model is available in Appendix A.

The model is trained using cross-entropy loss, which together with softmax has become the standard for CNN [Chen et al., 2021]. It also uses the Adam optimiser [Kingma and Ba, 2014] to update network weights iteratively based on training data.

A scheduling technique defined by `StepLR`, which decays the learning rate of each parameter group by a constant *gamma* every pre-defined *step* epochs, was also used.

Finally, the accuracy of the model on the test dataset gives us a quantifiable measure of how well our model performs in terms of classifying images into their respective categories.

Optimising hyperparameters like the learning rate of the optimiser, the batch size for dividing the data being input, the step size and gamma for scheduling, and the number of epochs is critical to the performance of the CNN model. For this purpose, we employed Optuna, an advanced hyperparameter optimisation library. Optuna automates the process of finding the most effective hyperparameters by running a series of trials, each time adjusting the parameters and assessing the model's performance. After 100 trials, the optimal values were determined and set as the default parameters for our CNN model. This systematic approach ensured that our model achieved a balance between learning efficiency and generalisation ability. The optimised values used for the final classifier are in the table below:

Epochs	Learning Rate	Batch Size	Step Size	Gamma
28	0.0025474653904364957	100	10	0.0815640608097732

Table 2: Optimised parameters for CNN

The final version of our CNN model, which incorporates the optimised hyperparameters, exhibits a significant improvement in performance and efficiency. It was saved for future use, allowing for quick deployment in image classification tasks.

4 Result and Analysis

The trained classifiers were run on the testing data and performance metrics were computed. These results are presented and discussed below.

4.1 Presentation of results

The performance metrics we have chosen to present are accuracy, precision, recall, and F1 score. In the multi-class case, all the metrics, apart from accuracy, are computed for each class and then averaged to get the overall metric for the classifier as a whole. The intuition behind each metric is:

1. Accuracy: $\frac{TP+TN}{TP+TN+FP+FN}$ - the proportion of correct predictions out of all the predictions,
2. Precision: $\frac{TP}{TP+FP}$ - e.g. proportion of correctly predicted shirts among all the predicted shirts,
3. Recall: $\frac{TP}{TP+FN}$ - e.g. proportion of the correctly predicted shirts among all the true shirts,
4. F1 - score: $2 * \frac{Precision*Recall}{Precision+Recall}$ - a harmonic mean of precision and recall.

In Table 3 we have shown the performance metrics for each of our models.

Metric	SVM	Bayes	CNN
Accuracy	0.8508	0.7160	0.8848
F1 Score	0.8513	0.7154	0.8850
Precision	0.8520	0.7225	0.8853
Recall	0.8508	0.7160	0.8848

Table 3: Performance Metrics for Classifiers

To further assess the performance of the classifiers within each class, we found their confusion matrices and computed each classifier's F1 score for each class.

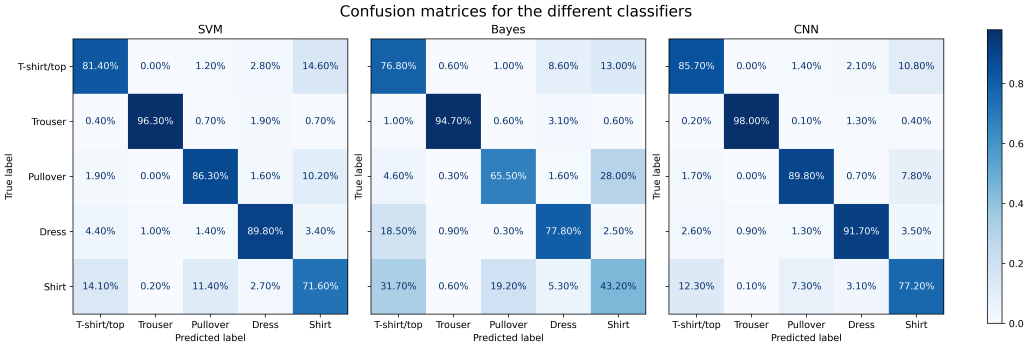

Figure 7: Confusion Matrices for the SVM, Bayes, and CNN classifiers

Class	SVM	Bayes	CNN
T-shirt/Top	0.8051	0.6604	0.8550
Trouser	0.9752	0.9609	0.9835
Pullover	0.8587	0.7020	0.9040
Dress	0.9034	0.7923	0.9308
Shirt	0.7142	0.4613	0.7791

Table 4: F1 score of the classifiers for each class

4.2 Classifier comparison and interpretation

The performance metrics outlined previously in Table 3 establish a clear hierarchy in terms of classifier efficacy. The CNN stands out for its superior performance across all the metrics compared to Naive Bayes and SVM. This could be because CNN is more specifically designed to work with image analysis than SVM or Naive bayes. CNN is able to extract more complex patterns in the images by creating a hierarchy of features. This makes it exceptionally suitable for image classification applications. From the presented metrics we can see that SVM is still a viable classifier as its performance metrics outperform Bayes and comes close to the CNN. This could be because there are non-linear patterns present in the data which SVM is able to capture using non-linear kernel functions, while Bayes is only able to make strictly linear decision boundaries. It is also important to note that the difference in performance could be due to the different features used for each classifier. CNN and SVM both used the scaled pixel features, while Naive Bayes used the first 2 linear discriminant variables as the features. The lower performance of Naive Bayes could thus be connected to the loss of information caused by the reduced dimensionality.

An examination of the confusion matrices in Figure 7 offers a detailed view of how each classifier performs in differentiating between the various classes of the dataset. These matrices together with the F1 scores provided in Table 4 help understand the nature of misclassifications made by each model. We can see that all the classifiers have the lowest F1 score for the Shirt class, indicating that all the models have difficulty differentiating between Shirts and other classes. This is likely due to the Shirt class being very similar to the other classes thus having a lot of overlap and being harder to separate. For example, we can see in the confusion matrix for Bayes - Figure 7 middle plot - that only 43.20% of the observations in the Shirt class were predicted correctly. The rest were misclassified mainly as the Pullover class, with 28.0%, or the T-shirt/top class, with 13%. On the other hand, the classifiers all had the largest F1 score for the Trouser class, indicating that they are best at discriminating between the Trouser class and other classes. Intuitively, this seems reasonable since the Trouser class is the only class referring to clothing covering the lower-body and thus has the most different features.

5 Conclusion

In conclusion, our analysis and comparison of three distinct classifiers on the Fashion-MNIST dataset have provided valuable insights into their respective strengths and weaknesses. All the models performed adequately, however, CNN emerged as the top-performing model based on all the performance metrics. This is an expected outcome as CNN is specifically designed to work with image classification. Throughout our project, we also found that the choice of dimensionality reduction method can be an important factor when performing image classification. Comparing LDA and PCA, we can see that the supervised method, LDA, produces features that prioritise class separation, yielding clearer boundaries between classes.

While this project yielded satisfactory results and meaningful insights, we must acknowledge its limitations and possible improvements. Firstly, it should be noted that the comparative analysis of the classifiers is highly dependent on our dataset. Therefore, while CNN was the best classifier for our data, it can not be generalised to all datasets. Due to limited time and resources, some of the hyperparameters for the classifiers were based on previous research rather than optimised through cross-validation. Optimising the parameters could potentially improve the performance of the classifiers.

Furthermore, experimenting with the complexity of the classifiers could yield better performance. It would also be interesting to test out a wider range of classifiers as this could lead to finding an even better method for image classification.

6 References

[pyt, 2023] (2023). Pytorch documentation — pytorch master documentation.

[skl, 2023] (2023). scikit-learn 0.22.1 documentation.

[Alpaydin, 2014] Alpaydin, E. (2014). *Introduction to Machine Learning*. The MIT Press.

[Bridle, 1990] Bridle, J. S. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. *Neurocomputing*, pages 227–236.

[Chen et al., 2021] Chen, L., Li, S., Bai, Q., Yang, J., Jiang, S., and Miao, Y. (2021). Review of image classification algorithms based on convolutional neural networks. *Remote Sensing*, 13(22):4712.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *ArXiv.org*.

[Liu et al., 2009] Liu, B., Yang, Y., Webb, G., and Boughton, J. R. (2009). A comparative study of bandwidth choice in kernel density estimation for naive bayesian classification. In Theeramunkong, T., Kijsirikul, B., Cercone, N., and Ho, T.-B., editors, *Proceedings of the 13th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD 2009)*, volume 5476, pages 302–313. Springer-Verlag London Ltd.

[Xiao et al., 2017] Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *ArXiv:1708.07747 [Cs, Stat]*.

[Zahadat, 2023] Zahadat, P. (2023). Lecture 14: Support vector machines (svm).

7 Appendices

A CNN Model

Model Summary:

- Epochs: 28
- Learning rate: 0.0025474653904364957

Layers:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 28, 28]             320
              ReLU-2          [-1, 32, 28, 28]               0
         MaxPool2d-3          [-1, 32, 14, 14]               0
            Conv2d-4          [-1, 64, 14, 14]          18,496
              ReLU-5          [-1, 64, 14, 14]               0
         MaxPool2d-6            [-1, 64, 7, 7]               0
           Dropout-7            [-1, 64, 7, 7]               0
           Flatten-8                [-1, 3136]               0
            Linear-9                 [-1, 128]         401,536
             ReLU-10                 [-1, 128]               0
          Dropout-11                 [-1, 128]               0
           Linear-12                   [-1, 5]             645
================================================================
Total params: 420,997
Trainable params: 420,997
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.70
Params size (MB): 1.61
Estimated Total Size (MB): 2.31
----------------------------------------------------------------
```

Criterion:

- CrossEntropyLoss()

B Derivation of the dual representation for SVC

Primal representation of the optimisation problem:

$$\min_{(w,b)} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{subject to constraints:} \quad y_i(w \cdot x_i + b) \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0$$

Lagrangian function of the primal representation with KKT conditions:

$$L(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i}^{n}\alpha_i[y_i(w \cdot x_i + b) - 1]$$

with the KKT conditions being:

$$\begin{aligned}
&\alpha_i \geq 0 \\
&y_i(w \cdot x_i + b) - 1 \geq 0 \\
&\alpha_i[y_i(w \cdot x_i + b) - 1] = 0
\end{aligned}$$

Partial derivatives with respect to both w and b:

$$\frac{dL}{dw} = w - \sum_{i}^{n}\alpha_i y_i x_i$$

and

$$\frac{dL}{db} = 0 - \sum_{i}^{n}\alpha_i y_i$$

Replacing w with $w = \sum_{i=1}^{N}\alpha_i y_i x_i$ and using $\sum_{i=1}^{N}\alpha_i y_i = 0$:

$$\begin{aligned}
L(w, b, \alpha) &= \frac{1}{2}||w||^2 - \sum_{i}^{n}\alpha_i[y_i(w \cdot x_i + b) - 1] \\
&= \frac{1}{2}\sum_{i=1}^{N}\alpha_i y_i x_i \cdot \sum_{j=1}^{N}\alpha_j y_j x_j - \sum_{i}^{n}\alpha_i\left[y_i(\sum_{j=1}^{N}\alpha_j y_j x_j \cdot x_i + b) - 1\right] \\
&= \frac{1}{2}\sum_{i=1}^{N}\alpha_i y_i x_i \cdot \sum_{j=1}^{N}\alpha_j y_j x_j - \sum_{i}^{n}\alpha_i y_i(\sum_{j=1}^{N}\alpha_j y_j x_j \cdot x_i + b) + \sum_{i}^{n}\alpha_i \\
&= \frac{1}{2}\sum_{i=1}^{N}\alpha_i y_i x_i \cdot \sum_{j=1}^{N}\alpha_j y_j x_j - \sum_{i}^{n}\alpha_i y_i \sum_{j=1}^{N}\alpha_j y_j x_j \cdot x_i - \sum_{i}^{n}\alpha_i y_i b + \sum_{i}^{n}\alpha_i \\
&= \frac{1}{2}\sum_{i=1}^{N}\alpha_i y_i x_i \cdot \sum_{j=1}^{N}\alpha_j y_j x_j - \sum_{i}^{n}\alpha_i y_i x_i \cdot \sum_{j=1}^{N}\alpha_j y_j x_j - \sum_{i}^{n}\alpha_i y_i b + \sum_{i}^{n}\alpha_i \\
&= \frac{1}{2}\sum_{i=1}^{N}\alpha_i y_i x_i \cdot \sum_{j=1}^{N}\alpha_j y_j x_j - \sum_{i}^{n}\alpha_i y_i x_i \cdot \sum_{j=1}^{N}\alpha_j y_j x_j + \sum_{i}^{n}\alpha_i = \\
&= \sum_{i}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j(x_i \cdot x_j)
\end{aligned}$$

Dual Representation:

$$\max_{\alpha}\left(\sum_{i}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j(x_i \cdot x_j)\right) \quad \text{subject to } 0 \leq \alpha \leq C \text{ and } \sum_{i}^{n}\alpha_i y_i = 0$$